On the Analysis of Mutation Operators in Multiobjective Cartesian Genetic Programming for Designing Combinational Logic Circuits

Lucas Augusto Müller de Souza , Heder Soares Bernardino

[1] Universidade Federal de Juiz de Fora (UFJF) – Juiz de Fora – MG – Brazil

`{lucasmuller,heder}@ice.ufjf.br`

Abstract. *Approximate Computing is an emerging paradigm that takes advantage of inherently error resilient digital circuits to design circuits with higher energetic efficiency, lower delay, or a smaller occupied area on the chips. Traditional approaches do not handle multiple objectives and metaheuristics appear as a proper alternative. In particular, Multiobjective Cartesian Genetic Programming (MOCGP) can find good solutions to the design and optimization of approximate circuits. The performance of CGP depends on the mutation adopted, as normally CGP only uses mutation for creating new solutions. However, to the best of our knowledge, just the traditional point mutation (PM) was used by the previously proposed MOCGP. Thus, the literature lacks an analysis of the best mutation operators of MOCGP. We propose here the analysis of PM and Single Active Mutation(SAM) on the multiobjective optimization of 15 heterogeneous combinational logic circuits from scratch and starting with a feasible solution. The results indicate that SAM obtained better results than PM.*

1. Introduction

Nowadays, electronic systems are present in the daily life of society. These systems base part of their operation on digital circuits that are usually designed to fully implement the logic behavior of some specifications (such as the truth table). However, there are some inherently error resilient applications where some logic behavior can be approximated. These are part of the Approximate Computing (AC) paradigm, where computational systems are allowed to tolerate a loss of accuracy so that circuits can be designed with higher energetic efficiency, a slower response time (i.e. delay), or a lower occupied area on the chips. The design of ACs is useful in practical situations, such as arithmetic circuits (e.g. adders and multipliers) [Hrbacek et al. 2016, Lima et al. 2019], multimedia systems [Oliveira et al. 2015] and, more recently, to improve the accuracy and hardware efficiency of neural networks [Ansari et al. 2019, Mrazek et al. 2020].

The trade-off between multiple objectives on the design of ACs can be studied using Evolutionary Algorithms (EAs). This type of technique obtained competitive solutions when compared to traditional approaches for designing combinational logic circuits (CLCs). Also, EAs solve the optimization problem without the knowledge of specific rules used by experts by testing combinations that the specialists would not be able to.

The initial efforts aimed to design logical circuits were made in [Koza 1992] using Genetic Programming. A general form of Genetic Programming called Cartesian Genetic Programming (CGP) was proposed in [Miller and Thomson 2000]. CGP can encode a wide range of programs represented as Direct Acyclic Graphs (DAGs) arranged in a 2-dimensional grid of nodes. Later, an in deep discussion on characteristics, applications, and variants of CGP can be found in [Miller 2011].

Initially, CGP used a single-objective to design digital circuits. However, the industry aims to design circuits energetically efficient, smaller, faster, with lower complexity and still have good precision. One can see that these objectives are conflicting. For instance, the complexity of the circuit may increase to reduce errors. Thus, using Multiobjective Cartesian Genetic Programming (MOCGP) is a proper alternative. An evolutionary approach to design approximate circuits that starts with fully functional circuits is proposed in [Hrbacek et al. 2016], where three objectives are considered: error, power dissipation, and delay. A combination of CGP and the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [Deb et al. 2002] is proposed to obtain approximate adders and multipliers. Although NSGA-II is a widely used technique for solving multiobjective optimization, its computational complexity depends on the population size. Therefore, a technique to design approximate circuits based on CGP with an Adaptive Population Size (APS) is presented in [Lima et al. 2019]. This technique was used to design and optimize 8-bit adders, 8-bit multipliers, and Arithmetic Logic Units (ALUs). Despite the good results found using CGP when solving the multiobjective optimization of combinational logic circuits, the previous studies adopted the standard point mutation.

As the offspring in CGP are usually generated using mutation operators only, the proper selection of this operator is important to the success of the method. Thus, we propose here an analysis of PM [Miller 2011] and SAM [Goldman and Punch 2013] on the design and optimization of approximate circuits. These operators are combined with NSGA-II and APS to obtain the best approach to design approximate circuits with multiple objectives. Also, we analyze the initialization of the search technique using the standard random approach and a feasible candidate design. These techniques are applied to the design of 15 heterogeneous circuits of a benchmark from [de Souza et al. 2020]. This collection of problems contains a more comprehensive set of circuits than those used on literature [Hrbacek et al. 2016, Lima et al. 2019].

2. Cartesian Genetic Programming (CGP)

Cartesian Genetic Programming (CGP) [Miller and Thomson 2000] is a type of Genetic Programming (GP) in which candidate solutions are encoded as Directed Acyclic Graphs (DAGs) represented as a 2-dimensional grid of nodes. The number of columns (n_c) and number of rows(n_r) of this grid are user-defined parameters. Each node has a fixed number of inputs (N_i) and outputs (N_o) and contains information on its inputs and function. These nodes can be connected to a primary input, i.e. input of the program, or another node in the columns on its left side in the grid. The number of columns allowed to be considered as input of a given node is also a user-defined parameter denoted by levels-back (l_b). Although the genotype has a fixed-length, the phenotype length is variable according to the number of active nodes, i.e. those that are connected (even non-directly) to any output of the program.

The most adopted search technique for Single-objective CGP is (1+λ)-Evolutionary Strategy [Miller 2011], where λ is the number of offspring generated at each generation, and the symbol $+$ indicates that the fittest candidate solution between the offspring and the parent is promoted to the next generation. The offspring in CGP are usually generated using mutation operators and the most common is Point Mutation (PM) [Miller 2011]. Another operator widely adopted in the literature is Single Active Mutation (SAM) [Goldman and Punch 2013].

Although CGP presents competitive results [Hrbacek et al. 2016, Miller 2011] for designing CLCs, there are some gaps between the research field of using EAs and the needs of the industry [Vasicek 2018]. The scalability is the main issue, as the processing time for solving the optimization problem depends on the size of the truth table and its number of rows grows exponentially based on the number of inputs.

2.1. Mutation Operators

As previously discussed, usually the only genetic operator used by CGP to generate the offspring is the mutation. So, the proper selection of this operator is important to the success of the method. In the literature, a wide range of mutation operators were proposed and the most common ones are point mutation [Hrbacek et al. 2016, Lima et al. 2019, Miller 2011, da Silva et al. 2019, de Souza et al. 2020, Vasicek and Sekanina 2014] and single active mutation [Goldman and Punch 2013, da Silva et al. 2019, de Souza et al. 2020].

The standard mutation operator in CGP is **Point Mutation (PM)**, where a gene value is randomly changed to another valid one. The modifications can occur in a node's input gene (according to the l_b defined), a node's function gene, or an output gene. The number of genes mutated can be defined by a percentage (μ_r) of the total number of nodes.

The **Single Active Mutation (SAM)** [Goldman and Punch 2013] operator also randomly modifies a gene to another valid value, similar to PM. However, to reduce the number of wasted evaluations, the chromosome is changed until at least one active node is changed. This ensures the existence of phenotypic differences.

3. Multiobjective Cartesian Genetic Programming (MOCGP)

Initially, CGP was proposed by Miller *et al.* [Miller and Thomson 2000] to design digital circuits considering a single objective. However, the industry needs to optimize circuits regarding multiple objectives. Thus, several algorithms were proposed to optimize digital circuits considering more than one objective.

One of the most famous is the *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II), proposed by Deb [Deb et al. 2002]. The combination of CGP and NSGA-II was capable of achieving good results and was proposed by Hrbacek *et al.* [Hrbacek et al. 2016]. Whereas, Lima *et al.* [Lima et al. 2019] proposed another method with an adaptive population size that obtained better results than CGP-NSGAII.

Despite the good results obtained by these methods, PM was used as a mutation operator in both cases. We analyze here the use of both PM and SAM as a mutation operator, as SAM was capable of finding better results than PM in single-objective optimization [Goldman and Punch 2013, de Souza et al. 2020].

3.1. Objective Functions

On the design of digital circuits as multiobjective, there are four objectives often considered: area, delay, power dissipation, and error. These objectives represent the industries need as they allow us to design smaller thus cheaper circuits, faster ones, with high energetic efficiency and still have good accuracy (i.e. low error rate). Here, we used the same objectives used by Lima *et al.* [Lima et al. 2019] and adopted the same definitions.

Error. There are many different metrics to measure the error of a given circuit when compared to the specified truth table, such as mean absolute error, relative error, the worst-case error, and hamming distance, with the last one being the most used [Lima et al. 2019, da Silva et al. 2019, de Souza et al. 2020]. Instead of using a circuit's truth table, the hamming distance can be calculated using Binary Decision Diagrams (BDD). Vasicek *et al.* [Vasicek and Sekanina 2014] proposed the use of BDD to speed up the feasibility evaluation of candidate solutions and reduce the scalability issue. BDD is a DAG containing one root and two terminal nodes that are referred to as 0 and 1. All the other nodes are called decision nodes or non-terminal. Each one of these nodes is a Boolean value and has two successors, the low (0) and the high (1) child. Every path on a BDD lead to one of the terminal nodes and each variable appears at most once. Considering a candidate circuit (C_c) and a feasible circuit (F_c) with outputs $o_1, ..., o_m$ and $o'_1, ..., o'_m$, respectively, BDD can be used to check the equivalency of these circuits. This equivalency occurs when, for each output i, the result of r_i = o_i XOR $o'_i = 0$ (false).

Propagation Delay. The propagation delay of a digital circuit is calculated as the delay of the longest path as follows Delay(C) $= \max_{\forall p \in \text{path}} \sum_{c_i \in p} t_d(c_i)$ where t_d is the delay of a cell c_i.

Power Dissipation. To estimate the power consumption we used the approach based on its switching component of power($P_{switching}$), short-circuit component of power ($P_{short-circuit}$) and the leakage current of power ($P_{leakage}$). The power dissipation is calculated as $P = a_{0\to 1} \cdot C_L \cdot f_{clk} \cdot V_{dd}^2 + I_{sc} \cdot V_{dd} + I_{leakage} \cdot V_{dd}$, where C_L is the load capacitance, f_{clk} is the clock frequency, I_{sc} is the short-circuit current, $I_{leakage}$ is the leakage current, V_{dd} is the supply voltage and $a_{0\to 1}$ is the node activity factor which quantifies the average number of times a logic gate makes a state transition that dissipates power within a period of clock and it can be defined as $a_{0\to 1} = p_0 \cdot p_1 = p_0 \cdot (1 - p_0)$.

3.2. Non-Dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II [Deb et al. 2002] is one of the most famous Multiobjective Evolutionary Algorithm. It is based on the *Pareto dominance* idea and the solutions are ranked in non-dominated fronts. The population P_t of size N and the offspring Q_t, generated using genetic operators, are combined as $R_t = P_t \cup Q_t$ and R_t is organized in non-dominated fronts. The first front (F_0) contains the current non-dominated solutions. The solutions in F_0 are removed from R_t and the second front (F_1) is created. This process is repeated until every solution is assigned to its corresponding front of dominance. A constraint handling procedure is necessary when the problem contains constraints. In NSGA-II, a given solution S_i dominates another solution S_j if: (i) S_i is feasible and S_j is not, (ii) S_i and S_j are both infeasible, but S_i has a small overall constraint violation or (iii) S_i and S_j are feasible and S_i dominates S_j as previously defined.

The population replacement in NSGA-II finds the smallest i such that $|F_0 \cup \cdots \cup F_i| = M_i \geq N$, where $|\cdot|$ denotes the cardinality of a set. $P_{t+1} = F_0 \cup \cdots \cup F_i$ when $M_i = N$. Otherwise, $F_0 \cup \cdots \cup F_{i-1}$ and $N - |F_0 \cup \cdots \cup F_{i-1}|$ individuals in F_i compose P_{t+1}. Those individuals of F_i with the largest crowding distance are selected to the next

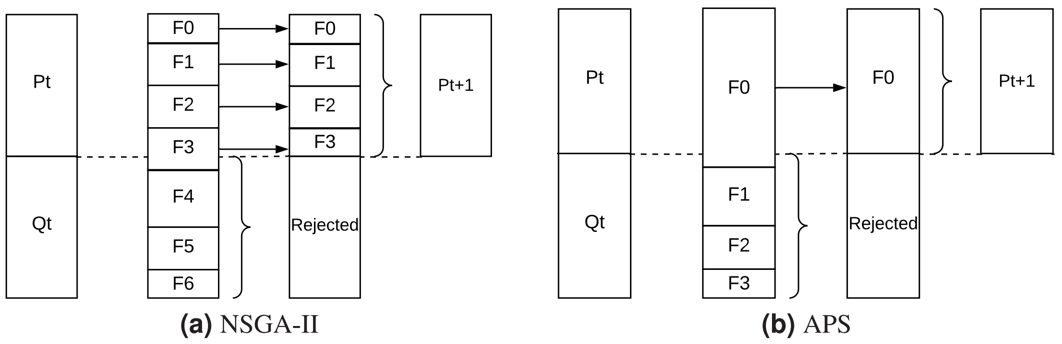

Figure 1. Population replacement, where P_t represents the parents of generation t and Q_t is the offspring. The dashed lines represents the population size or the maximum population size (m_{ps}), respectively, for NSGA-II and APS.

generation. The crowding distance of an individual is half of the perimeter of a hypercube formed by its nearest neighbors, or infinity for the extreme solutions (those with the best objective function values). The replacement of NSGA-II is illustrated in Figure 1a.

The combination of NSGA-II with CGP to design digital circuits was proposed in [Hrbacek et al. 2016] and was capable of evolving 8-bit adders and multipliers with significant power consumption savings. Hence, we used that method on the computational experiments to study the impact of the mutation operators on the design of digital circuits.

3.3. Adaptive Population Size (APS)

The computational complexity of NSGA-II depends on the population size. So, even if a larger population increases the coverage of the search space, it also increases the computational cost of the method. Otherwise, a small population implies a lower computational cost but it can lead to premature convergence.

To address that problem, APS was proposed in [Lima et al. 2019] to adapt the population size during the search process. Instead of selecting a fixed number of individuals, that may come from other candidate solutions than the non-dominated ones, APS selects only the non-dominated individuals. This is important to obtain a wide coverage of the search space whereas the computational cost remains lower than with fixed population size. One can notice that the population (and consequently the computational cost required in each generation) increases with the number of non-dominated solutions. Thus, a parameter ($\max_{ps}$) was included in APS to control the maximum population size and to limit the budget spent in each iteration. This process may result in two situations: (i) the number of non-dominated solutions in the current population is smaller than or equal to $\max_{ps}$ and, therefore, all these individuals are promoted to the next generation, or (ii) the crowding distance is calculated for the current non-dominated solutions and those with the largest values are selected. Figure 1b illustrates this process.

4. Computational Experiments

There are some mutation operators in the literature for the design of combinational logic circuits, such as PM [Miller 2011], GAM [da Silva et al. 2019] and SAM [Goldman and Punch 2013]. Also, SAM normally achieved results better than these other approaches when a single objective is considered. Meanwhile, in the new research

Table 1. The selected test-problems, including its name, number of inputs (In.), number of outputs (Out.), the maximum number of evaluations of the individuals (Eval.) available to the techniques, the objective function values of the baseline circuit when seeding the population with conventional architecture and its functionality.

Group	Name	In.	Out.	Eval.	Baseline			Functionality
					Error	Delay(ns)	Power(mW)	
1	C17	5	2	1.20E+7	0.00	11.30	0.188048	Logic
	cm42a	4	10	1.28E+7	0.00	13.00	0.960585	Logic
	cm82a	5	3	8.00E+6	0.00	28.30	0.811869	Logic
	cm138a	6	8	1.92E+7	0.00	16.40	0.730218	Logic
	decod	5	16	1.20E+7	0.00	11.30	0.448242	Logic
	f51m	8	8	1.92E+7	0.00	50.40	2.63543	Arithmetic
	majority	5	1	8.00E+6	0.00	10.20	0.127364	Voter
	z4ml	7	4	1.68E+7	0.00	57.20	2.14865	2-bit Adder
2	9symml	9	1	2.16E+7	0.00	157.50	3.39376	Count Ones
	alu2	10	6	1.60E+7	0.00	126.90	5.06011	ALU
	cm85a	11	3	1.76E+7	0.00	43.60	1.67714	Logic
	cm151a	12	2	2.88E+7	0.00	23.20	0.7064	Logic
	cm162a	14	5	4.48E+7	0.00	24.90	1.00049	Logic
	cu	14	11	4.48E+7	0.00	26.60	0.781757	Logic
	x2	10	7	2.40E+7	0.00	21.50	0.878563	Logic

area known as Approximate Computing with Multiobjective CGP, APS and NSGA-II are methods from the literature able to optimize complex architectures such as Arithmetic Logic Units, and 8-bits adders and multipliers. However, there is still a lack of analysis on the impact of the mutation operator in MOCGP as the previous works only used PM. Thus, we investigate the impact of using SAM and PM with APS and NSGA-II.

To evaluate the association of these mutation and selection approaches, we used the small (group 1) and medium (group 2) problems from the benchmark proposed in [de Souza et al. 2020]. This benchmark was created to evaluate single-objective metaheuristics in the design of digital circuits and contains a set of heterogeneous problems. The large problems (group 3) and alu4, which belong to group 2, have been disregarded due to computational resources limitation.

The circuits can be designed from scratch when the initial solution is randomly generated (labeled by R). Also, a feasible solution can be used as the initial candidate circuit, for instance, using ESPRESSO as in [de Souza et al. 2020] (labeled by E). The computational experiments consider 2 multiobjective approaches (APS and NSGA-II), 2 mutation operators (PM and SAM), and two strategies for generating the initial solution (R and E). As a result, 8 techniques are analyzed here, and they are labeled according to these three features: approach-mutation-initialization. The comparative analyses are initially performed considering the same initialization strategy.

The parameters of CGP are $n_r = 1$, $n_c = 100$ for problems in Group 1, and $n_c = 1000$ for those in Group 2, and function set $\Gamma = \{$AND, OR, NOT, NAND, NOR, XOR,XNOR$\}$, such as in [de Souza et al. 2020], and the levels-back used was $l_b = n_c$ and the population size was 50. The parameters t_d, V_{dd} and C_L (Section 3.1) used to evaluate the delay and power dissipation are the same used in [Lima et al. 2019]. These values were obtained from the Nexperia[1] manufacturer and the logic gates are: 74LVC1G08 (AND), 74LVC1G32 (OR), 74AHC1G04 (NOT), 74LVC1G00 (NAND), 74LVC1G02 (NOR), 74LVC1G86 (XOR) and HEF4077B (XNOR). The error was calculated using the BuDDy Package[2], which is an implementation of BDD. The feasible circuit used as a base to be compared with the candidate solution is that provided in [de Souza et al. 2020]. To ensure that only solutions with a low acceptable error is generated, we constrained the mean relative error to be at most 10% as Hrbacek *et al.* [Hrbacek et al. 2016]. The source-code and the raw data of the experiments are available[3].

4.1. Performance Assessment

The hypervolume indicator, introduced in [Zitzler and Thiele 1998], is considered a fair performance measure as solutions (i) closer to the optimal Pareto front and (ii) well-distributed along the whole front have a higher hypervolume indicator value. This indicator uses the non-dominated solutions and a reference point (usually, the worst values of the objective functions) to calculate the hypervolume. An improved algorithm to calculate the hypervolume indicator was proposed in [Fonseca et al. 2006] and its implementation[4] was used here to evaluate the performance of the solutions.

Statistical tests were also used to investigate the similarity of the results obtained by the techniques. The Dunn p-values were calculated using all results obtained for each problem separately. The null hypothesis is considered rejected here when p-value ≤ 0.05.

4.2. Optimization from scratch

The first considered scenario in the proposed investigation is the design of CLCs from scratch, i.e. the population is randomly initialized. Tables 2 and 3 present the results obtained. A star (*) is used to identify the techniques that obtained results that are statistically different from those that found the best median (highlighted in boldface) according to Dunn's test.

Considering the 8 problems in group 1, NSGAII-PM-R and APS-PM-R obtained the best median hypervolume or are statistically similar to the best ones in all problems, while NSGAII-SAM-R and APS-SAM-R reached this performance in 5 and 4 problems, respectively. For the problems in group 2, NSGAII-SAM-R, APS-SAM-R, and NSGAII-PM-R obtained the best median results or their results are statistically similar to the best ones, respectively, in 7, 6, and 1 problem. Thus, PM performed better than SAM in the small problems (those in group 1), whereas SAM performed better than PM in medium ones (those in group 2).

NSGAII-SAM-R obtained $SR = 100\%$ in all problems (i.e. at least one solution does not violate the constraint in any independent run), whereas APS-SAM-R, NSGAII-PM-R, and APS-PM-R obtained $SR = 96.00\%$, $SR = 72.00\%$ and $SR = 76.00\%$ respectively, for 1 problem. In general, the techniques using SAM obtained a higher success rate than those using PM, and the best performing search technique is NSGAII-SAM-R due to its best median hypervolume values in 12 of 15 problems and obtained $SR = 100\%$ in all problems.

[1]`https://www.nexperia.com/`

[2]`https://sourceforge.net/projects/buddy`

[3]`https://github.com/ciml/eniac2020_mocgp`

[4]`http://lopez-ibanez.eu/hypervolume`

Table 2. Results obtained when starting with a random population – group 1.

Problem	Method	Best	Median	Worst	Mean	Std	SR(%)
C17	NSGAII-PM-R	0.6622	0.6339	0.6106	0.056	0.4758	**100.0**
	NSGAII-SAM-R+	**0.6626**	**0.6449**	**0.6219**	0.0615	0.418	**100.0**
	APS-PM-R	0.6579	0.6253	0.601	0.0707	**0.4088**	**100.0**
	APS-SAM-R	0.662	0.6289	0.5979	**0.0809**	0.4271	**100.0**
cm42a	NSGAII-PM-R	0.7396	0.6884	0.6931	0.0246	0.638	**100.0**
	NSGAII-SAM-R*	0.7364	0.6776	0.6785	0.0329	**0.5871**	**100.0**
	APS-PM-R	**0.7701**	**0.7068**	**0.6993**	**0.0342**	0.6306	**100.0**
	APS-SAM-R*	0.7456	0.6892	0.6823	0.0321	0.6261	**100.0**
cm82a	NSGAII-PM-R	0.6566	**0.5906**	**0.5788**	0.0676	0.3802	**100.0**
	NSGAII-SAM-R+	**0.7081**	0.5839	0.578	**0.0975**	**0.1987**	**100.0**
	APS-PM-R	0.6743	0.5888	0.5765	0.0614	0.447	**100.0**
	APS-SAM-R	0.7036	0.5489	0.5492	0.0892	0.3127	**100.0**
cm138a	NSGAII-PM-R	0.8319	**0.8009**	**0.802**	0.0188	0.7769	**100.0**
	NSGAII-SAM-R*+	0.8231	0.7798	0.7893	0.0183	0.757	**100.0**
	APS-PM-R	0.8293	0.7966	0.774	**0.1267**	**0.1742**	**100.0**
	APS-SAM-R	**0.8381**	0.7831	0.7936	0.0204	0.7706	**100.0**
decod	NSGAII-PM-R	0.7668	**0.7507**	**0.7427**	0.0174	0.7089	**100.0**
	NSGAII-SAM-R+	0.7625	0.7347	0.7333	0.0207	0.6902	**100.0**
	APS-PM-R	0.7847	0.7468	0.7418	**0.0252**	0.69	**100.0**
	APS-SAM-R*	**0.7877**	0.7227	0.7273	0.023	**0.6896**	**100.0**
f51m	NSGAII-PM-R	**0.6002**	0.3544	**0.3486**	**0.1057**	**0.0374**	**100.0**
	NSGAII-SAM-R+	0.5295	0.3478	0.348	0.0957	0.1535	**100.0**
	APS-PM-R	0.4542	**0.3619**	0.3277	0.0972	0.1204	**100.0**
	APS-SAM-R	0.5222	0.3214	0.3249	0.0927	0.1612	**100.0**
majority	NSGAII-PM-R	0.7293	0.6214	0.6034	0.0779	0.4331	**100.0**
	NSGAII-SAM-R*+	0.7537	0.4989	0.5154	0.1179	**0.3233**	**100.0**
	APS-PM-R	**0.7846**	**0.631**	**0.6335**	0.085	0.4548	**100.0**
	APS-SAM-R*	0.7685	0.5094	0.5413	**0.1183**	0.3457	**100.0**
z4ml	NSGAII-PM-R	0.8931	0.8	0.7492	0.1311	0.4067	**100.0**
	NSGAII-SAM-R	0.8965	0.7624	0.7323	0.1568	0.3436	**100.0**
	APS-PM-R	0.8878	**0.8435**	**0.7825**	0.1445	0.2434	**100.0**
	APS-SAM-R*	**0.9057**	0.6715	0.6477	**0.2506**	**0.0046**	**100.0**

4.3. Optimization using a feasible circuit

The second scenario is the optimization of CLCs with a feasible circuit in the initial population, as in [de Souza et al. 2020]. Tables 4 and 5 present the results obtained by the techniques, where a star (*) indicates the statistical differences with respect to the best median results and the best values are highlighted using boldface. In this scenario, the techniques using SAM (NSGAII-SAM-E and APS-SAM-E) obtained the best median hypervolume results or are statistically similar to the best ones, respectively, in 8 and 7 problems of the 8 problems in group 1. On the other hand, the methods with PM (NSGAII-PM-E and APS-PM-E) obtained the best results in 5 and 3 problems, respectively. Considering the 7 problems in group 2, NSGAII-SAM-E and APS-SAM-E obtained the best results or are statistically similar to the best ones in all of them. NSGAII-PM-E and APS-PM-E reached no result statistically similar to the best ones. All techniques obtained $SR = 100.00\%$ in all problems considered here.

Table 3. Results obtained when starting with a random population – group 2.

Problem	Method	Best	Median	Worst	Mean	Std	SR(%)
9symml	NSGAII-PM-R*	0.6444	0.3871	0.3988	0.1066	0.2228	**100.0**
	NSGAII-SAM-R	**0.7709**	**0.5864**	**0.5548**	0.1216	0.2828	**100.0**
	APS-PM-R*	0.6324	0.4436	0.473	0.1016	0.3166	**100.0**
	APS-SAM-R*	0.6828	0.4717	0.4607	**0.1593**	**0.0459**	**100.0**
alu2	NSGAII-PM-R*	0.184	0.0752	0.0706	0.0613	**0.0**	72.0
	NSGAII-SAM-R+	0.327	**0.2062**	**0.2068**	0.0628	0.1011	**100.0**
	APS-PM-R*	0.1636	0.0558	0.0612	0.0534	**0.0**	76.0
	APS-SAM-R	**0.3425**	0.1753	0.189	**0.1022**	**0.0**	96.0
cm85a	NSGAII-PM-R	0.7849	0.6974	0.6957	0.0398	0.6115	**100.0**
	NSGAII-SAM-R+	0.7747	**0.7049**	**0.7026**	0.0369	0.6283	**100.0**
	APS-PM-R*	0.7438	0.6771	0.6789	0.038	**0.5722**	**100.0**
	APS-SAM-R	**0.8029**	0.6903	0.6943	**0.0408**	0.6288	**100.0**
cm151a	NSGAII-PM-R*	0.8392	0.6653	0.6443	**0.1199**	**0.4008**	**100.0**
	NSGAII-SAM-R	**0.8991**	0.7749	0.7694	0.0946	0.5574	**100.0**
	APS-PM-R*	0.8284	0.6749	0.6708	0.084	0.4841	**100.0**
	APS-SAM-R	0.8973	**0.8225**	**0.7911**	0.0879	0.561	**100.0**
cm162a	NSGAII-PM-R*	0.8192	0.7243	0.7095	**0.0801**	**0.5611**	**100.0**
	NSGAII-SAM-R	0.8648	**0.7899**	0.7632	0.0755	0.6164	**100.0**
	APS-PM-R*	0.8365	0.7209	0.7233	0.0691	0.5849	**100.0**
	APS-SAM-R	**0.8733**	0.7871	**0.7715**	0.0586	0.6101	**100.0**
cu	NSGAII-PM-R*	0.8904	0.8697	0.855	0.0404	0.7595	**100.0**
	NSGAII-SAM-R+	0.9166	**0.8919**	**0.8769**	0.0356	0.769	**100.0**
	APS-PM-R*	0.8914	0.869	0.8323	**0.0602**	**0.7084**	**100.0**
	APS-SAM-R	**0.9199**	0.8903	0.8718	0.0484	0.7595	**100.0**
x2	NSGAII-PM-R*	0.7677	0.7129	0.7094	**0.0424**	**0.5944**	**100.0**
	NSGAII-SAM-R+	0.7869	**0.7582**	**0.7458**	0.0384	0.6763	**100.0**
	APS-PM-R*	0.7705	0.722	0.7213	0.0329	0.6566	**100.0**
	APS-SAM-R	**0.7898**	0.7509	0.7455	0.0325	0.6706	**100.0**

In general, the techniques using SAM are statistically better than those using PM for problems in both groups considered. The best performing approach is NSGAII-SAM-E, due to its best median hypervolume values in all problems.

4.4. Analysis of the Results of NSGAII-SAM-R and NSGAII-SAM-E

Here we compare the results obtained by the best technique in each scenario, namely NSGAII-SAM-R (solutions are optimized from scratch) and NSGAII-SAM-E (the initial population contains a feasible solution). The median hypervolume values obtained by these techniques were used in the comparisons. Also, the Dunn-test was performed for all problems and a (+) is used to highlight the method that is statistically different from the best one, between the results obtained by NSGAII-SAM-R and NSGAII-SAM-E.

Considering the problems in group 1, NSGAII-SAM-E and NSGAII-SAM-R obtained the best result or its results are statistically similar to the best ones in 7 and 2 problems, respectively. Also, NSGAII-SAM-E obtained the best results or its results are statistically similar to the best ones in all the 7 problems in group 2, while NSGAII-SAM-R reached such performance only in 3 problems. Thus, as NSGAII-SAM-E obtained the

Table 4. Results obtained when starting with a feasible population – group 1.

Problem	Method	Best	Median	Worst	Mean	Std	SR(%)
C17	NSGAII-PM-E	0.6564	0.5696	0.5431	0.0851	0.4263	**100.0**
	NSGAII-SAM-E+	**0.6613**	0.605	0.5539	**0.1005**	0.4156	**100.0**
	APS-PM-E	0.6605	**0.6245**	**0.5837**	0.0765	0.4309	**100.0**
	APS-SAM-E	0.6608	0.5748	0.5546	0.0866	**0.4149**	**100.0**
cm42a	NSGAII-PM-E	**0.7921**	0.7174	0.7182	0.0298	0.6637	**100.0**
	NSGAII-SAM-E	0.7691	0.7168	0.7226	0.023	0.6895	**100.0**
	APS-PM-E*	0.7625	0.7058	0.7048	**0.0308**	**0.6463**	**100.0**
	APS-SAM-E	0.7504	**0.7252**	**0.7228**	0.0147	0.6827	**100.0**
cm82a	NSGAII-PM-E	0.7326	0.6412	0.6355	0.0478	0.5367	**100.0**
	NSGAII-SAM-E	0.7122	**0.6599**	**0.6482**	0.0449	0.5318	**100.0**
	APS-PM-E*	0.6722	0.6306	0.6249	0.0387	0.5366	**100.0**
	APS-SAM-E	**0.7337**	0.6397	0.6387	**0.0664**	**0.4523**	**100.0**
cm138a	NSGAII-PM-E	0.8329	0.805	0.8081	**0.0118**	0.7919	**100.0**
	NSGAII-SAM-E	0.8267	0.8105	0.8124	0.0069	0.7975	**100.0**
	APS-PM-E*	0.8313	0.8049	0.8046	0.0096	**0.7877**	**100.0**
	APS-SAM-E	**0.8422**	**0.8118**	**0.8132**	0.0097	0.7932	**100.0**
decod	NSGAII-PM-E*	0.8357	0.8224	0.8197	**0.0109**	0.7981	**100.0**
	NSGAII-SAM-E	**0.8437**	**0.8272**	**0.8258**	0.0098	0.7969	**100.0**
	APS-PM-E	0.8322	0.8249	0.822	0.0104	**0.7837**	**100.0**
	APS-SAM-E	0.8372	0.8261	0.825	0.0085	0.8006	**100.0**
f51m	NSGAII-PM-E*	0.3719	0.2993	0.3022	**0.0248**	**0.26**	**100.0**
	NSGAII-SAM-E	0.7638	0.7109	0.7138	0.021	0.6812	**100.0**
	APS-PM-E*	0.3781	0.3156	0.3189	0.0212	0.2785	**100.0**
	APS-SAM-E	**0.8003**	**0.7148**	**0.7161**	0.0234	0.6679	**100.0**
majority	NSGAII-PM-E	0.6683	0.5928	0.5991	0.0278	0.5295	**100.0**
	NSGAII-SAM-E	**0.7094**	0.6021	0.5974	**0.0346**	**0.5159**	**100.0**
	APS-PM-E	0.6342	**0.6078**	**0.6051**	0.0221	0.5759	**100.0**
	APS-SAM-E*	0.6325	0.5836	0.5873	0.0264	0.5309	**100.0**
z4ml	NSGAII-PM-E*	0.6779	0.3496	0.3756	0.0772	0.3151	**100.0**
	NSGAII-SAM-E	0.9003	**0.7873**	**0.8043**	0.0591	0.7053	**100.0**
	APS-PM-E*	0.7987	0.3509	0.4229	**0.15**	**0.3036**	**100.0**
	APS-SAM-E	**0.9011**	0.7775	0.7897	0.0485	0.7006	**100.0**

best results in most of the problems considered here (14 of 15) when designing ACs.

5. Concluding Remarks and Future Works

We analyzed the use of both Point Mutation (PM) and Single Active Mutation (SAM) on the design of approximate circuits with Cartesian Genetic Programming (CGP). These operators were combined with NSGA-II and APS, an adaptive population size approach. The techniques were used to design 15 heterogeneous circuits in two scenarios, where the search process is initialized (i) from scratch and (ii) with a fully functional circuit.

In the first scenario, NSGAII-SAM-R obtained the best median results and, in the second one, NSGAII-SAM-E found the best median results. In both scenarios, SAM obtained better results than PM, as in single-objective optimization problems. Using CGP with SAM, the replacement strategy of NSGA-II, and a feasible circuit in the initial popu-

Table 5. Results obtained when starting with a feasible population – group 2.

Problem	Method	Best	Median	Worst	Mean	Std	SR(%)
9symml	NSGAII-PM-E*	0.5974	0.4011	0.3317	**0.2057**	**0.0095**	**100.0**
	NSGAII-SAM-E	0.6661	0.5813	**0.592**	0.0306	0.5399	**100.0**
	APS-PM-E*	0.0299	0.0208	0.0203	0.004	0.0151	**100.0**
	APS-SAM-E	**0.6804**	**0.5858**	0.5906	0.0463	0.4565	**100.0**
alu2	NSGAII-PM-E*	0.2841	0.2385	0.2392	0.0139	**0.2112**	**100.0**
	NSGAII-SAM-E	**0.6989**	**0.6606**	0.6578	0.0225	0.5955	**100.0**
	APS-PM-E*	0.2879	0.2492	0.2489	0.019	0.2181	**100.0**
	APS-SAM-E	0.6984	0.6585	**0.6587**	**0.025**	0.6109	**100.0**
cm85a	NSGAII-PM-E*	0.7801	0.7554	0.755	0.0104	0.7312	**100.0**
	NSGAII-SAM-E	**0.8915**	0.8788	0.8774	0.0084	0.8551	**100.0**
	APS-PM-E*	0.7743	0.7516	0.7513	**0.0112**	**0.7253**	**100.0**
	APS-SAM-E	0.8882	**0.8804**	**0.8786**	0.0066	0.8631	**100.0**
cm151a	NSGAII-PM-E*	0.8322	0.7762	0.7488	0.0665	0.6192	**100.0**
	NSGAII-SAM-E	0.8945	**0.8229**	**0.8202**	0.0555	0.6913	**100.0**
	APS-PM-E*	0.8407	0.752	0.7336	**0.0728**	**0.6012**	**100.0**
	APS-SAM-E	**0.906**	0.8227	0.8188	0.0547	0.7042	**100.0**
cm162a	NSGAII-PM-E*	0.7454	0.702	0.6941	0.0289	0.6163	**100.0**
	NSGAII-SAM-E	**0.8212**	0.7557	0.7624	0.0263	0.7168	**100.0**
	APS-PM-E*	0.7462	0.6856	0.6896	**0.0337**	**0.6059**	**100.0**
	APS-SAM-E	0.8181	**0.7761**	**0.7775**	0.0214	0.7335	**100.0**
cu	NSGAII-PM-E*	0.8847	0.8603	0.8545	**0.0279**	**0.757**	**100.0**
	NSGAII-SAM-E	**0.9269**	0.9172	**0.9166**	0.0067	0.8915	**100.0**
	APS-PM-E*	0.8944	0.8707	0.8611	0.0252	0.7884	**100.0**
	APS-SAM-E	0.9261	**0.9183**	0.9153	0.0139	0.8522	**100.0**
x2	NSGAII-PM-E*	0.781	0.7476	0.7375	0.0369	**0.6464**	**100.0**
	NSGAII-SAM-E	**0.8432**	**0.7884**	0.7858	0.0271	0.73	**100.0**
	APS-PM-E*	0.7987	0.7494	0.7431	**0.0396**	0.647	**100.0**
	APS-SAM-E	0.8414	0.7882	**0.7879**	0.0227	0.7418	**100.0**

lation is the combination that reached the best results in most problems of both scenarios considered here.

The combination of NSGA-II and SAM to design other approximate circuits, such as adders, multipliers, and ALUs is an interesting research avenue. Also, we intend to include the number of transistors as an objective.

Acknowledgments

The authors thank the financial support provided by CNPq (312682/2018-2), FAPEMIG (APQ-00337-18), and UFJF. Also, this work has been supported by UFJF's High-Speed Integrated Research Network (RePesq).

References

Ansari, M. S., Mrazek, V., Cockburn, B. F., Sekanina, L., Vasicek, Z., and Han, J. (2019). Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

da Silva, J. E. H., de Souza, L. A., and Bernardino, H. S. (2019). Cartesian genetic programming with guided and single active mutations for designing combinational logic circuits. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 396–408. Springer.

de Souza, L. A. M., da Silva, J. E. H., Chaves, L. J., and Bernardino, H. S. (2020). A benchmark suite for designing combinational logic circuits via metaheuristics. *Applied Soft Computing*, page 106246.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evol. Computation*, 6(2):182–197.

Fonseca, C. M., Paquete, L., and López-Ibánez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *2006 IEEE international conference on evolutionary computation*, pages 1157–1163. IEEE.

Goldman, B. W. and Punch, W. F. (2013). Reducing wasted evaluations in cartesian genetic programming. In *European Conference on Genetic Programming*, pages 61–72. Springer.

Hrbacek, R., Mrazek, V., and Vasicek, Z. (2016). Automatic design of approximate circuits by means of multi-objective evolutionary algorithms. In *Intl. Conf. on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6. IEEE.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Lima, L. S., Bernardino, H. S., and Barbosa, H. J. (2019). Designing combinational circuits using a multi-objective cartesian genetic programming with adaptive population size. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 592–604. Springer.

Miller, J. F. (2011). Cartesian genetic programming. In *Cartesian Genetic Programming*, pages 17–34. Springer.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. *Proc. of EuroGP 2000*, 1802:121–132.

Mrazek, V., Sekanina, L., and Vasicek, Z. (2020). Using libraries of approximate circuits in design of hardware accelerators of deep neural networks. In *Proc. of the Intl. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*, pages 243–247. IEEE.

Oliveira, J. R., Soares, L. B., Costa, E., and Bampi, S. (2015). Energy-efficient gaussian filter for image processing using approximate adder circuits. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 450–453. IEEE.

Vasicek, Z. (2018). Bridging the gap between evolvable hardware and industry using cartesian genetic programming. In *Inspired by Nature*, pages 39–55. Springer.

Vasicek, Z. and Sekanina, L. (2014). How to evolve complex combinational circuits from scratch? In *Proc. of the Intl. Conf. on Evolvable Systems*, pages 133–140. IEEE.

Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer.